Eötvös Loránd Tudományegyetem

Természettudományi Kar

Futárszolgálat modellezése

Készítette:

Nagy Eszter

Tóth Benjámin

2022

Tartalomjegyzék:

Tartalomjegyzék:
2

A projekt
3

A probléma modellezése
4

- Úthálózat
  4
- Éttermek
  5
- Megrendelők és futárok
  6
- Rendelések
  7
- Közlekedési eszközök
  8

Optimális párosítás megtalálása
8

- Mátrix konstruálása Dijkstra-algoritmus segítségével
  9
- Optimalizálás
  10

Kapott eredmény megjelenítése
12

- Utasítások a megfelelő futároknak
  12
- Grafikus megjelenítés
  13
- Adott futár útvonala
  16

Zárszó
17

Köszönet
17

A projekt

Egy futárcég néhány problémáját vettük kiindulásul a projektünkhöz. Olyan cégekre kell gondolni, mint a Wolt[1], illetve a Foodpanda[2]. Mindkét cég egy úgynevezett online ételrendelési portált működtet. A rendszerben összegyűjtik a házhozszállítást vállaló éttermeket, és a vásárlók számára egy kényelmes felületet nyújtanak, melyen keresztül a kiválasztott étteremnek leadhatják rendelésüket. Az elkészített rendelést nem az étterem saját futárja kézbesíti, hanem a futárcég küld egy futárt a kézbesítendő szállítmányért. Minden futár a cég színeiben közlekedik, de különböző futárokat különbözőképpen kezel a rendszer. Természetesen figyelembe veszi, hogy milyen közlekedési eszközt használ a futár. Leggyakrabban biciklit használnak, de autós, motoros és gyalogos futárok is dolgozhatnak. Olyan cég is van, ami a korábban elvégzett feladatok mennyiségét illetve minőségét is figyelembe veszi. Ezek alapján kapnak a futárok személyre szabott megbízásokat.
Beszélgettünk az egyikünk ismerősével, aki a Wolt-nál dolgozik biciklis futárként. Kaptunk egy kis betekintést a rendszerükbe, és abba a felületbe amit egy ott dolgozó futár lát. Megtudtuk, hogy egy futár körülbelül mekkora távolságokat szokott címenként megtenni, és hogy mennyi időt ad a rendszer egy feladat teljesítésére, illetve azt is megtudtuk, hogy nagyon ritkán visznek egyszerre több rendelést. Hasznos információ volt számunkra az is, hogy habár a rendszer nagyon is figyelembe veszi az adott futár helyzetét, az útvonalat mégsem szabja meg. Egy futár csupán azt a két címet kapja meg, ahol a rendelést át kell vennie és ahova kézbesítenie kell.
Ezeken alapszik a mi projektünk. Nem egy online algoritmuson kezdtünk el dolgozni, hanem egy feladatkiosztós problémán, ami egy adott pillanatban a lehető legoptimálisabban oszt ki megbízásokat futároknak. Célunk volt, hogy a modellünk több tényezőt figyelembe tudjon venni, mint amit a futárcégek is figyelembe vesznek, illetve látványosan szerettük volna megjeleníteni egy ilyen probléma egy optimális megoldását. Python nyelvet választottuk, és a Colab[3] felületén dolgoztunk egy .ipynb formátumú notebook file-ban.

---

[1] https://wolt.com/hu/hun/budapest

[2] https://www.foodpanda.hu/en/?r=1

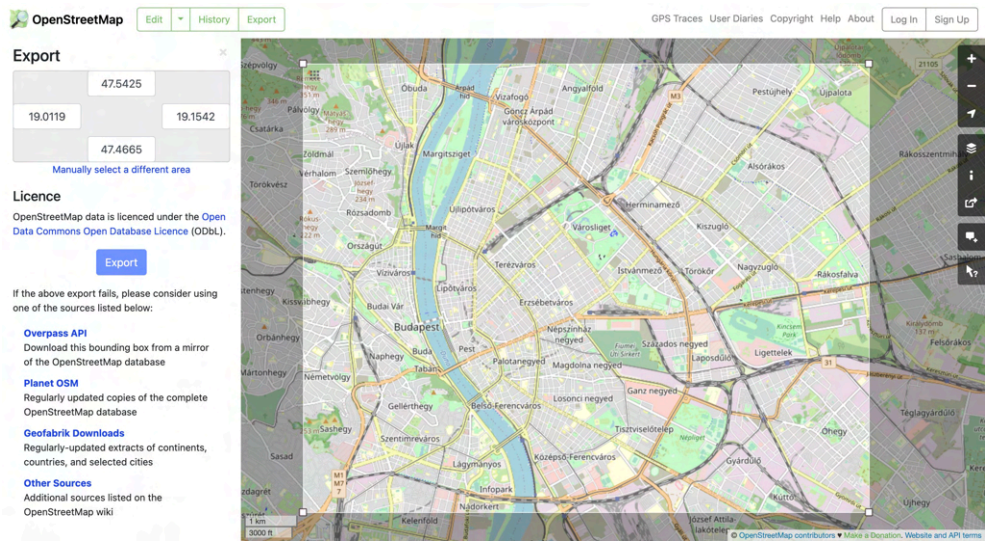
[3] Google Colaboratory

A probléma modellezése

- **Úthálózat**

A modellünk csontvázát az úthálózat képezi, hiszen a futárok, az éttermek és a megrendelők is az úthálózat mentén helyezkednek el, illetve azon mozogva juthatnak el egyik helyről a másikra a futáraink.

Mi valós adatokkal szerettünk volna dolgozni, ezért a Google Maps adatait használtuk. Egy OpenStreetMap[4] nevű oldalról tudtunk letölteni ilyen adatokat egyszerűen, ahol egy térképen ki kellett jelölni egy téglalapot, ami lefedett egy nagyobb területet. Az adatbázis tartalmazta Budapest belvárosát, de egészen az ELTE lágymányosi campus-ától az Örs vezér teréig húzódott, észak fele pedig a teljes Margit-szigetet is tartalmazta. Az OpenStreetMap a Google Maps által tárolt információkat ezen területről kiimportálta egy .osm formátumú file-ba.



Ez a speciális file sokkal többet tudott, mint amire nekünk szükségünk volt. Nem csupán az úthálózat adatait tartalmazta, hanem mindenféle számunkra felesleges adatot közlekedési táblákról, az utak utolsó aszfaltozás időpontjáról, túraútvonalakról és ezer más tényezőről. Nekünk csak az utakra volt szükségünk, amiket a Google egészen gráfszerűen tárol. A gráf csúcsai 10-12 karakter hosszú számkódok, amikhez földrajzi koordináták tartoznak, és két csúcs között akkor fut él, ha az egyikből el lehet jutni a másikba. Úgy is fel lehet fogni, hogy a csúcsok az utcasarkok, és minden csúcs a szomszédos sarokhoz tartozó csúccsal van összekötve, azzal a kiegészítéssel, hogy csak egyeneseket tudunk tárolni, ezért a kanyarodó utak sok egymáshoz közeli csúccsal vannak megoldva, mint egy töröttvonal. Ezeket az adatokat tartottuk meg, és alakítottuk át az eredeti file-t három .txt file-ba.

[4] https://www.openstreetmap.org/

Az elsőben csúcsok szomszédságát tároltuk úgy, hogy azt majd a kódunk be tudja olvasni, és egy dictionary-t tudjon belőle készíteni, amiben a kulcsok a gráf csúcsai, a kulcsokhoz tartozó értékek pedig az adott csúcsok szomszédjainak halmaza. A másodikban éllistásan tároltuk őket, ahol csupán egy listában, élszámnyi kételemű tuple-ben tartunk minden olyan csúcspárt, amelyek között a gráfban fut él. A harmadikban szintén dictionary-t használunk, ahol az elsőhöz hasonlóan a kulcsok a csúcsok, viszont itt az értékek az adott csúcsokhoz tartozó földrajzi koordináták egy kételemű tuple-ben tárolva.

```
import ast
import names
import random
import pulp
import numpy as np
import pylab as pl
from pqdict import PQDict
from matplotlib import pyplot as plt
from matplotlib import collections  as mc
from numpy.ma.core import append
from matplotlib.pyplot import figure

file1 = open("n_coords")
file2 = open("n_edgelist")
file3 = open("n_neighbours")
data1 = file1.read()
data2 = file2.read()
data3 = file3.read()
file1.close()
file2.close()
file3.close()
coords = ast.literal_eval(data1)
neighbours = ast.literal_eval(data3)
edgelist =  ast.literal_eval(data2)
edgelist_set = set(edgelist)
nodes = list(coords.keys())
```

(A fentebbi tömb importálja a későbbiekben használt összes python package-et.)

- **Éttermek**

Összegyűjtöttük néhány általunk ismert budapesti étterem és kávézó címét, és a listát bővítettük pár fiktív hellyel is, hogy bevihessük azok adatait a kódunkba. Ezt egy dictionary-vel oldottuk meg, amiben a kulcsok a vendéglátó egységek nevei, az értékek pedig a hely koordinátái. Mivel az adott földrajzi koordinátához nem feltétlen tartozik csúcs, ezért a kódunk a hozzá legközelebb esőt használja fel és azt a csúcsot veszi az étterem helyének. Ez

nem okozott gondot, mivel nagyon sűrűn vannak csúcsok, ezért a kód mindig talált elég közelit. Habár a kódunk az általunk bevitt éttermekre fut, de ezek könnyen módosíthatóak vagy bővíthetőek más éttermekkel.

```
shopsinput = {
    "B.E.B.GEP": (47.50279219494247, 19.143080688997205),
    "B.E.B.DOH": (47.50027021972165, 19.07261156538511),
    "Artizan": (47.50394811572796, 19.05281002749366),
    "Eco Cafe" : (47.50763687918182, 19.066481011324413),
    "Tibiado Pékség" : (47.49526143173333, 19.062382290819397),
    "The Box": (47.50968066007557, 19.056794522613266),
    "Wekni Pékség" : (47.51553631060966, 19.055222427739523),
    "Réparetek" : (47.50523484557339, 19.08193440877897),
    "UMO Étterem" : (47.50169791036156, 19.038761573011406),
    "Ganesha Ételbár" : (47.47780286185868, 19.052580313696),
    "Vegakuckó Étkezde" : (47.48121089084858, 19.066956953826008),
    "Café Zsivágó" : (47.503218750433525, 19.061678098096134),
    "Freyja" : (47.49976686870447, 19.073685879015567),
    "Cafe Frei" : (47.47000745586774, 19.059511013889175),
    "Cserpes Tejivó" : (47.47509671622191, 19.048530696088733),
    "Fruccola Mom": (47.49045415633097, 19.022154419861362),
    "Cinnamon": (47.50809906347719, 19.034038592240513),
    "Budapest Garden": (47.53187078846506, 19.043163252362632),
    "Breadpit Pékség": (47.52481999288331, 19.12438003458586),
    "Hathi Indiai Kifőzde": (47.53619203572932, 19.073173171297103),
    "Taj Mahal Étterem": (47.51095019377097, 19.064568353377847),
    "Katica Kávéház": (47.50445308869438, 19.10149090843288),
    "Essence Delicates": (47.52254161679863, 19.1035040839082)
                }
shoplist = []
shops = {}
j = 0
for val in shopsinput.values():
    mindist = 10
    for i in range(len(nodes)):
        if mindist > ((coords[nodes[i]][0]-val[1])**2+((coords[nodes[i]]
[1]-val[0])**2))**0.5:
            mindist = ((coords[nodes[i]][0]-val[1])**2+((coords[nodes[i]]
[1]-val[0])**2))**0.5
            nthnode = i
    shops[nodes[nthnode]] = list(shopsinput.keys())[j]
    j += 1
shoplist = list(shops.keys())
```

- **Megrendelők és futárok**

Mivel a megrendelők és a futárok a valóságban sem fix pontok, ezért a Python beépített random függvényével sorsoltuk ki azok koordinátáit. Természetesen a program arról a

területről kap koordinátákat, amin a kódunk többi része is dolgozik. és az éttermekhez hasonlóan azt is a legközelebbi csúcsra tűzzük ki.

```
edwards = []
locations1 = []
for i in range(200):
    locations1.append([random.uniform(19.01,
19.157),random.uniform(47.466, 47.542)])
for l in locations1:
    mindist = 10
    for i in range(len(nodes)):
        if mindist > ((coords[nodes[i]][0]-l[0])**2+((coords[nodes[i]]
[1]-l[1])**2))**0.5:
            mindist = ((coords[nodes[i]][0]-l[0])**2+((coords[nodes[i]]
[1]-l[1])**2))**0.5
            nthnode = i
    edwards.append(nodes[nthnode])
customers = []
locations2 = []
for i in range(100):
    locations2.append([random.uniform(19.01,
19.157),random.uniform(47.466, 47.542)])
for l in locations2:
    mindist = 10
    for i in range(len(nodes)):
        if mindist > ((coords[nodes[i]][0]-l[0])**2+((coords[nodes[i]]
[1]-l[1])**2))**0.5:
            mindist = ((coords[nodes[i]][0]-l[0])**2+((coords[nodes[i]]
[1]-l[1])**2))**0.5
            nthnode = i
    customers.append(nodes[nthnode])
```

- **Rendelések**

A rendelések felfoghatóak úgy, mint megrendelő-étterem párok. A mi modellünkben kikötöttük, hogy minden megrendelő pontosan egy rendelést ad le, tehát pontosan egy párban szerepel. Nem jelentett volna gondot egy olyan megvalósítás sem, ahol több rendelést is ugyanahhoz a megrendelőhöz lehet társítani, de a valós futárcégeknél sem tud egy vásárló több mint egy helyről egyszerre rendelni.

A megrendelőkhöz viszont véletlenszerűen sorsoltunk éttermeket, így megengedve hogy több, vagy akár nulla megrendelés is érkezhessen egy étteremhez, vagyis azt nem tudjuk hogy melyik étterem hányszor szerepel a párosításokban. A mi konstrukciónkban minden ilyen párhoz egy kételemű listát rendeltünk, amiben az adott megrendelőhöz és az étteremhez tartozó csúcsok azonosítóját tartjuk.

Létrehoztunk még egy számlálót is, ami az egyes éttermekbe érkezett rendelések számát tárolja. Ehhez egyszerűen egy listát használtunk.

```
orders = []
for c in customers:
    orders.append([c, random.choice(shoplist)])
ordercount = []
for s in shoplist:
    counter = 0
    for o in orders:
        if s == o[1]:
            counter += 1
    ordercount.append(counter)
```

- **Közlekedési eszközök**

Fontosnak tartottuk, hogy megjelenjenek a kódunkban a futárok közlekedési eszközei is. Biciklis, autós és motoros futár kategóriákat különböztettünk meg. Dictionary-t használtunk amiben a kulcsok helyén szerepelnek a futárok, az értékek pedig sorsolt közlekedési eszközök. Véletlenszerűen sorsoltunk, de alkalmaztunk egy súlyozást, hogy valósabb legyen a járművek aránya, hiszen biciklis futár gyakrabban fordul elő, mint autós vagy motoros.

```
edwards_vehicles = {}
vehicle = random.choices(["bicycle", "motorcycle", "auto"], [7, 2, 1],
k=len(edwards))
vehicle.sort()
for e in range(len(edwards)):
    edwards_vehicles[edwards[e]] = vehicle[e]
num_bicycle = vehicle.count("bicycle")
num_motorcycle = vehicle.count("motorcycle")
num_auto = vehicle.count(“auto")
```

Optimális párosítás megtalálása

Akkor optimális a megoldásunk, ha a lehető leggyorsabban tudják teljesíteni a futárok a megbízásokat. Ehhez egy megfelelő futár-rendelés párosításra van szükségünk. Felfogható ez úgy is, hogy egy olyan teljes páros gráfban keresünk minimális súlyú, az egyik pontosztályt lefedő teljes párosítást, amiben a lefedett pontosztály csúcsai az éttermek, a másik pontosztály pedig a futárok, és az adott élnek a költségét a hozzá tartozó futár-étterem távolság adja meg. Azért elegendő csupán az étteremig nézni az útvonalakat, mivel onnan már egyértelmű a legjobb útvonal a kézbesítési címhez, akármelyik futár kapja meg a megbízást.

- **Mátrix konstruálása Dijkstra-algoritmus segítségével**

Egy olyan mátrixot hoztunk létre, amiben az oszlopok felelnek meg a futároknak, a sorok pedig az éttermeket reprezentálják. Minden oszlop-sor metszetben az adott futár-étterem távolság szerepel. Ezen értékek megtalálásához a Dijkstra-algoritmust[5] használtuk, amit egy függvényként valósítottunk meg.

Az algoritmus egy gráfban megadja minden csúcs legrövidebb távolságát egy kiválasztott *s* csúcstól, illetve minden csúcshoz a legrövidebb utat is *s*-be.

A Dijkstra előtt a szomszédságokat tároló dictionary-t bővítettük az élhosszokkal. Erre azért volt szükség, mert így a függvény bemenetének csupán erre a dictionary-re, illetve egy *s* csúcsra van szüksége. Visszakapunk a függvénytől két dictionary-t, mindkettőben a kulcsok a csúcsok, viszont amíg az elsőben az értékek *s*-ből az adott csúcsokba vezető legrövidebb utak hosszai, addig a másodikban a csúcshoz tartozó "ősök" azonosítói. Egy adott csúcs őse az a szomszédos csúcs, ami felé az *s*-be vezető legrövidebb úton el kell indulni. Ha ezeket ismerjük, akkor a segítségükkel megkaphatjuk a legrövidebb utakat az összes csúcsra.

A mátrixos konstrukcióhoz elegendő volt a legrövidebb utak hosszát lekérni, magukkal az utakkal a későbbiekben foglalkoztunk.

```
def dijkstra(G, start, end=None):
    inf = float('inf')
    D = {start: 0}
    Q = PQDict(D)
    P = {}
    U = set(G.keys())
    while U:
        (v, d) = Q.popitem()
        D[v] = d
        U.remove(v)
        if v == end: break
        for w in G[v]:
            if w in U:
                d = D[v] + G[v][w]
                if d < Q.get(w, inf):
                    Q[w] = d
                    P[w] = v
    return D, P
```

(GitHub.com-on talált függvényt alakítottunk át[6])

[5] https://en.wikipedia.org/wiki/dijkstra%27s_algorithm

[6] Eredeti link: https://github.com/joyrexus/Dijkstra/blob/master/Dijkstra.py

Ezen a ponton jelenítettük meg a futárok típusait. Beépítettük, hogy a mátrix készítésénél a kód vegye figyelembe a futárok közlekedési eszközeit. A biciklist vettük alapértelmezettnek, az autós és a motoros esetén a mátrix adott elemét megszoroztuk egy egynél kisebb számmal, így reprezentálva, hogy azok gyorsabban tudnak mozogni. Ez a szám az autós esetén 0.9 a motoros esetén pedig 0.8 lett.

```
rows, cols = len(shoplist), len(edwards)
M = [([0]*cols) for i in range(rows)]
for s in shoplist:
    dist, pred = dijkstra(neighbours_dist, s)
    for e in edwards:
        if edwards_vehicles[e] == "auto":
            M[shoplist.index(s)][edwards.index(e)] = dist[e] * 0.9
        if edwards_vehicles[e] == "bicycle":
            M[shoplist.index(s)][edwards.index(e)] = dist[e]
        if edwards_vehicles[e] == "motorcycle":
            M[shoplist.index(s)][edwards.index(e)] = dist[e] * 0.8
```

- **Optimalizálás**

Készítettünk egy optimalizáló függvényt, ami bemenetként kapja a mátrixot és azt a listát, ami az éttermekbe beérkező rendelések számát tartalmazza. Futása után visszaad egy listát, amiben az optimális párosításhoz tartozó oszlop-, sorindex párok szerepelnek.

A függvényen belül egy IP[7] feladatot fogalmaztunk meg.

A mátrixnak minden eleméhez tartozik egy bináris változó, amire igaz, hogy:

$$x_{ij} = \begin{cases} 1 \text{ ha az } i. \text{ étterembe a } j. \text{ futár szállít.} \\ 0 \text{ egyébként.} \end{cases}$$

Kétféle feltételt vezettünk be:

$$\sum_{i=1}^{n} x_{ij} \leq 1 \quad j = 1,2,...,m$$

$$\sum_{j=1}^{m} x_{ij} = b_i \quad i = 1,2,...,n$$

b_i jelöli az i. étterembe érkezett rendelések számát.

[7] Egészértékű programozás (angolul integer programming)

Az első feltétel biztosítja, hogy minden futár maximum egy darab megbízást kapjon, a második pedig, hogy az adott étterembe pontosan ugyanannyi futár menjen, ahány megrendelő adott le oda rendelést.

A célfüggvényünk pedig a következő volt:

$$\min\{\sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij}\, c_{ij}\}$$

c_{ij} jelöli a j. futár távolságát az i. étteremtől.

Ez minimalizálja nekünk a párosításunk "költségét", vagyis az alkalmazott futárok által megtett utat.

Az IP modellt a PuLP[8] könyvtár segítségével valósítottuk meg, ami kifejezetten lineáris programozási feladatok megoldására való.

```
def opt(m, b):
    M = m
    x_name = []
    for i in range(len(M)):
        for j in range(len(M[0])):
            var = str(i) + "_" + str(j)
            x_name.append(var)
    x = [pulp.LpVariable(x_name[i], lowBound = 0, upBound = 1) for i in
range(len(x_name)) ]
    prob = pulp.LpProblem("problem", pulp.LpMinimize)
    for i in range(len(M)):
        prob += sum(x[i*len(M[0]):(i+1)*len(M[0])]) == b[i]
    for i in range(len(M[0])):
        prob += sum(x[i::len(M[0])]) <= 1
    c = 0
    t = 0
    for i in range(len(M)):
        for j in range(len(M[0])):
            c += M[i][j] * x[t]
            t += 1
    prob += c
    status = prob.solve()
    l = []
    for i in range(len(x)):
        if int(pulp.value(x[i])) == 1:
            b = i % len(M[0])
            a = i // len(M[0])
            l.append([a, b])
    return(l)
```

[8] https://coin-or.github.io/pulp/

Kódunk végül háromelemű listákat készít. Egy lista egy rendelést reprezentál, amiben az adott futár azonosítója, a kiválasztott étterem azonosítója, illetve a megrendelő azonosítója szerepel.

```
pairs = opt(M, ordercount)

pair_list = []
for p in pairs:
    pair_list.append([shoplist[p[0]], edwards[p[1]]])
for p in pair_list:
    for o in orders:
        if p[0] == o[1]:
            p.append(o[0])
            orders.remove(o)
            break
```

Kapott eredmény megjelenítése

A megkapott futár-étterem-megrendelő hármasokat vizualizálni is szerettük volna, mivel a kapott három elemű listák az azonosítókkal nehezen értelmezhetőek. A kódunk helyes működésének ellenőrzéséhez is nagy segítség, ha megjelenítjük valahogy a megoldást.

- **Utasítások a megfelelő futároknak**

Írtunk egy egyszerű kódot, ami kiírja, hogy a megbízást kapott futárok melyik étteremhez mennek. Hogy látványosabb legyen, a futárok kaptak egy nevet is. Ehhez a names bővítményt használtuk, és egy dictionary-ban minden futárhoz egy nevet rendeltünk.

```
edward_names = {}
for e in edwards:
    edward_names[e] = names.get_first_name()

for p in pair_list:
    print(edward_names[p[1]], "goes to", shops[p[0]])
```

A kiírás így néz ki:

```
Tim goes to Café Zsivágó
William goes to Freyja
Amanda goes to Artizan
Elliot goes to Eco Cafe
Donna goes to Taj Mahal
:
:
```

- **Grafikus megjelenítés**

Két különböző kódot irtunk, ami meg tudja jeleníteni a megrendeléseket egy térképen.

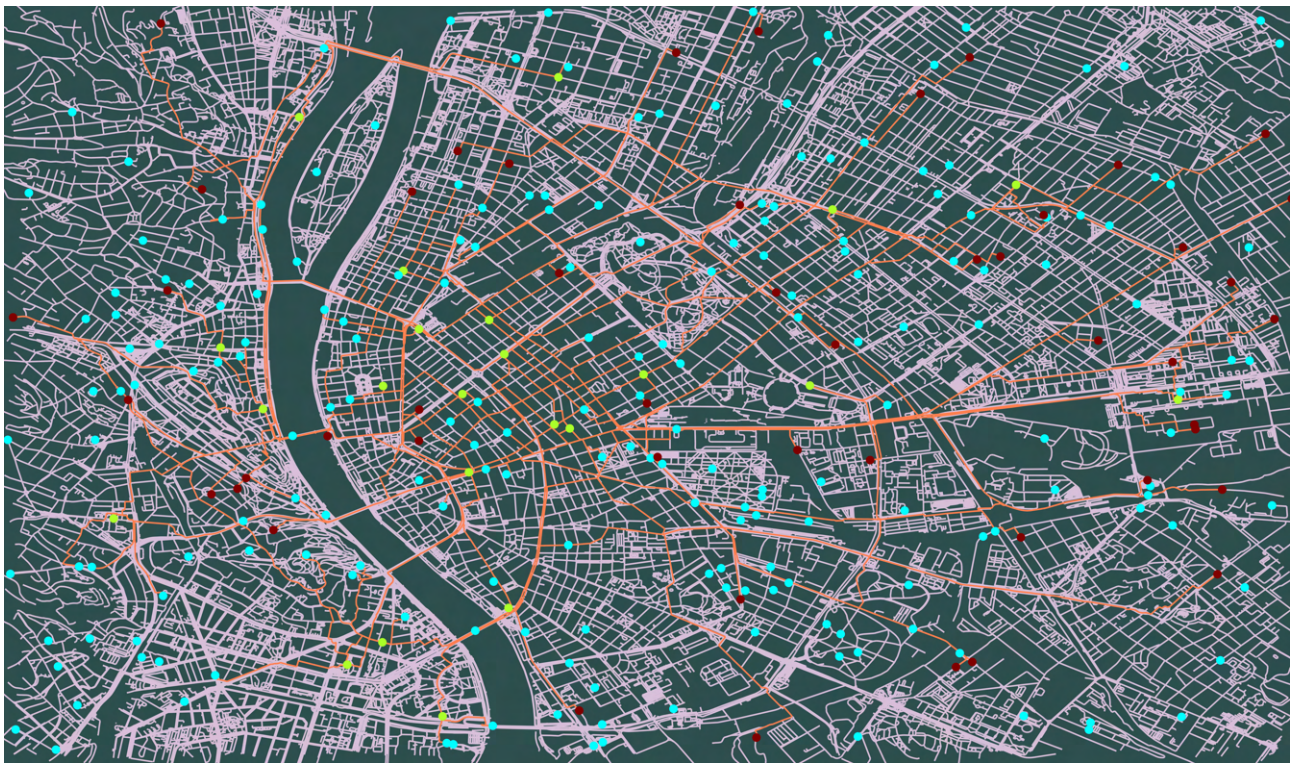
Az első egyszerűen kirajzol egy térképet arról a területről, ahol a futárok mozognak, majd megjeleníti rajta a futárokat, megrendelőket, és éttermeket is. Ezek után a program összeköti a futárokat azokkal az éttermekkel, ahova menniük kell.

kék = futár
piros = megrendelő
zöld = étterem
narancssárga = futár-étterem pár

Nagyon sokminden leolvasható egy ilyen képről. Szépen kirajzolódik Budapest úthálózata. Ha eléggé belenagyítunk, akkor láthatjuk, hogy az íves utak tényleg törött vonallal vannak megoldva. Látszólag eléggé szépen működik a véletlenszerű elhelyezése a futároknak és megrendelőknek. Leolvashatjuk, hogy melyik étteremről hányan rendeltek, jelen esetben például látunk olyan éttermet, ahova nem érkezett rendelés, egy másikból viszont öten is rendeltek.

A futár-étterem párokkal kapcsolatban fontos megjegyezni, hogy habár a narancssárga összekötő szakaszok légvonalban mennek, a távolságokat a program mégis tudja az utak mentén kalkulálni.

A második megjelenítésre használt kód abban különbözik az elsőtől, hogy ott narancssárgával a futárok útvonala van ábrázolva. Az útvonal a kiválasztott futároktól ezen esetben nem csupán az éttermekig tart, hanem egészen a megrendelőkig.



A második kód tartalmazza az elsőt, ezért nem térünk ki külön mindkettőre. A Matplotlib[9] könyvtárat használtuk a megjelenítéshez. Az egész kód pontok és szakaszok ábrázolását végzi a kezdeti gráfkonstrukció alapján, amiknek a helyzetét a koordinátákat tartalmazó dictionary mutatja. Az információt, hogy melyik útszakaszokat kell narancssárgán kirajzolnunk, a Dijkstra-algoritmus szolgáltatja az éttermektől nézett legrövidebb út keresésével. Visszaad egy csúcslistát az adott futárokig és megrendelőkig, ami mentén haladva kell alkalmaznunk a színezést.

```
path_list = []
help_list = [[0]]
for p in pair_list:
    if p[0] == help_list[-1][0]:
        help_list[-1].append(p[1])
        help_list[-1].append(p[2])
    else:
        help_list.append([p[0]])
        help_list[-1].append(p[1])
        help_list[-1].append(p[2])
help_list.pop(0)
for h in help_list:
    dist, pred = dijkstra(neighbours_dist, h[0])
    for i in range(1, len(h)):
        destination = h[i]
        to_path_list = [destination]
        while destination != h[0]:
            destination = pred[destination]
            to_path_list.append(destination)
        path_list.append(to_path_list)
```

[9] https://matplotlib.org/

```
xaed = []
yaed = []
xbed = []
ybed = []
xmed = []
ymed = []
for i in range(len(edwards)):
    if edwards_vehicles[edwards[i]] == "auto":
        xaed.append(coords[edwards[i]][0])
        yaed.append(coords[edwards[i]][1])
    if edwards_vehicles[edwards[i]] == "bicycle":
        xbed.append(coords[edwards[i]][0])
        ybed.append(coords[edwards[i]][1])
    if edwards_vehicles[edwards[i]] == "motorcycle":
        xmed.append(coords[edwards[i]][0])
        ymed.append(coords[edwards[i]][1])
xcu = []
ycu = []
for i in range(len(customers)):
    xcu.append(coords[customers[i]][0])
    ycu.append(coords[customers[i]][1])
xsh = []
ysh = []
for i in range(len(shoplist)):
    xsh.append(coords[shoplist[i]][0])
    ysh.append(coords[shoplist[i]][1])
lines = []
lines2 = []
for i in range(len(edgelist)):
    lines.append([coords[edgelist[i][0]], coords[edgelist[i][1]]])
for i in range(len(path_list)):
    for j in range(len(path_list[i])-1):
        lines2.append([coords[path_list[i][j]], coords[path_list[i]
[j+1]]])
lc = mc.LineCollection(lines, colors="thistle", linewidths=1)
lc2 = mc.LineCollection(lines2, colors="coral", linewidths=1)
fig, ax = plt.subplots(figsize=(18, 12), dpi=300)
ax.add_collection(lc)
ax.add_collection(lc2)
ax.autoscale()
ax.set_facecolor("darkslategray")
plt.scatter(xsh, ysh, s=20, color="greenyellow", zorder=10)
plt.scatter(xcu, ycu, s=20, color="maroon", zorder=10)
plt.scatter(xaed, yaed, s=20, color="mediumblue", zorder=10)
plt.scatter(xbed, ybed, s=20, color="aqua", zorder=10)
plt.scatter(xmed, ymed, s=20, color="deepskyblue", zorder=10)
```

- **Adott futár útvonala**

Írtunk még egy függvényt, ami egy választott futár útvonalát rajzolja ki. A bemenete a futár-étterem-megrendelő hármasok alkotó listák egy tetszőleges eleme.

Nagyon hasonlóan működik, mint a korábban bemutatott ábrázoló kódok:

```
def path_for_single_edward(e):
    shop = e[0]
    edward = e[1]
    address = e[2]
    path_list = []
    path_list.append(path(edward, shop))
    path_list.append(path(shop, address))
    lines = []
    lines2 = []
    for i in range(len(edgelist)):
        lines.append([coords[edgelist[i][0]], coords[edgelist[i][1]]])
    for i in range(len(path_list[0])-1):
        lines2.append([coords[path_list[0][i]], coords[path_list[0][i+1]]])
    for i in range(len(path_list[1])-1):
        lines2.append([coords[path_list[1][i]], coords[path_list[1][i+1]]])
    lc = mc.LineCollection(lines, colors="thistle", linewidths=1)
    lc2 = mc.LineCollection(lines2, colors="coral", linewidths=1)
    fig, ax = plt.subplots(figsize=(18, 12), dpi=300)
    ax.add_collection(lc)
    ax.add_collection(lc2)
    ax.autoscale()
    ax.set_facecolor("darkslategray")
    plt.plot(coords[edward][0], coords[edward][1], marker="o", color="aqua")
    plt.plot(coords[shop][0], coords[shop][1], marker="o", color="greenyellow")
    plt.plot(coords[address][0], coords[address][1], marker="o", color="maroon")
```

A kapott kép:

Zárszó

Az általunk megírt program nem a futárcégek által használt programokhoz hasonló megoldásokat alkalmaz. Az általános megközelítés valamilyen mohó algoritmus szokott lenni, ami tud folyamatosan működni. Ezzel szemben a mi programunk egy adott pillanatban tudja a korábban beérkezett rendeléseket a megfelelő futárokhoz rendelni, viszont ezt a feladatot hibátlanul elvégzi. Megtalálja azt a legjobb leosztást, amivel a lehető legkisebb össztávot mennek a futárok, ezzel egy olcsó és gyors megoldást szolgáltatva. Ha szeretnénk egy futárszolgálatnak biztosítani a folyamatos működést ezzel a kóddal, akkor azt úgy tehetjük meg, hogy bizonyos időközökkel futtatjuk a programot. Az adott futással, az előző futás óta beérkezett rendeléseknek kell futárokat találni, tehát az éppen kézbesítést végző futárokat figyelmen kívül kell hagyni.

Köszönet

Köszönjük Gergely Edvárdnak hogy időt szánt ránk és ismertette velünk a Wolt működését, tiszteletére a kódunkban “edward” néven hivatkoztunk a futárokra. Köszönjük Wágner Mariannának és Tóth Gábor Attilának az írásos munkánk lektorálását.